APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: USING PAGING TO INITIALIZE SYSTEM MEMORY

APPLICANT: JORDAN L. JUSTEN

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EV399289337US

November 21, 2003

Date of Deposit

USING PAGING TO INITIALIZE SYSTEM MEMORY

BACKGROUND

[0001] When a computer or other data processing system is first booted up, the operating system is not available to provide instructions for system operation. In the time interval between power on and operating system start up, firmware provides instructions to initialize the system.

[0002] System firmware includes instructions necessary to load portions of the operating system from a storage medium such as a hard disk into the system memory, as well as other instructions and data for initializing system components such as the main memory. Firmware is stored in a non-volatile medium such as flash memory. Flash memory is programmable non-volatile memory (i.e. it retains the stored information when the system is powered down but may be reprogrammed if desired).

[0003] System firmware is typically memory mapped. That is, a flash memory (or other) chip storing the firmware includes a hardware mechanism so that the instructions and data included in the firmware can be accessed via a standard processor memory read. Devices that are directly readable by the processor (memory mapped) are generally more expensive than devices which are not memory mapped.

DESCRIPTION OF DRAWINGS

[0004] FIG. 1 shows an implementation of a method for enabling the use of cache memory as RAM during system startup.

[0005] FIG. 2 is a schematic of a system including memory mapped and non-memory mapped firmware, according to some implementations.

[0006] FIG. 3 shows an implementation of a method for executing system firmware.

[0007] Like reference symbols in the various drawings indicate like elements.

DETAILED DESCRIPTION

[0008] Systems and techniques described herein may provide for more cost efficient data processing systems utilizing both memory mapped and non-memory mapped firmware. Additionally, the efficiency of a startup process may be improved by implementing a paging mechanism in at least a portion of system cache memory that is configured to function as RAM during system startup.

[0009] In some existing computer systems, the microprocessor uses cache memory to enhance performance. Rather than accessing the system memory each time it needs information, the microprocessor may initially access the system memory to retrieve data or instructions, then store

the data or instructions in the cache memory for future use. Since it is statistically more likely that the same information is repeatedly accessed, storing the information locally in a fast memory may provide a significant advantage.

[0010] Cache memory, which may be integrated with a processor or separate from the processor, is used as a performance enhancing feature during regular system operation. However, it has been recognized that system cache may be used prior to initialization of the main memory to enable more efficient system start up. Moreover, since the cache would ordinarily be unused during system startup, using system cache may allow the additional efficiency to be obtained without the cost or complexity of additional memory. Although implementations of the current systems and techniques utilize the cache memory, other implementations are possible (e.g., different and/or additional memory may be used).

[0011] FIG. 1 shows a method 100 in which system cache is used to enable more efficient startup. The cache may first be initialized so that system firmware has the ability to use the cache (in a small region of the system memory map) as readable and writeable memory (110). This process may be referred to as initializing the cache as RAM (CAR)

state. The initialization process may include requesting data for a range of memory addresses, which may not correspond to actual memory in the system. In response to these requests, the system may be configured so that placeholder data is returned (so that the processor will not hang). Once data has been returned for a range of addresses to be used for the cache, subsequent reading of the data corresponding to that address range will be from the cache rather than from other parts of the system.

[0012] The cache may then be partitioned (120). For example, the cache may be partitioned to a region for data, a region for pages, and (optionally) a region for page tables. In implementations using software-managed translation lookaside buffer (TLB), page tables may be omitted. Paging may then be enabled in the processor (130), and the firmware may be executed (140).

[0013] As noted above, firmware provides data and instructions to be used prior to execution of the operating system. In existing systems, firmware is typically memory mapped and includes the instructions necessary for system startup. Firmware execution according to current systems and techniques may be implemented using a system 200 as shown in FIG. 2 and a method shown in FIG. 3. System 200 includes firmware having two parts: memory mapped firmware

210, as well as non-memory mapped firmware 220. Memory mapped firmware 210 may be stored on a medium such as a flash memory chip including a hardware mechanism allowing direct reading by the processor. Memory mapped firmware is configured to initialize CAR and paging.

[0014] Non-memory mapped firmware 220 may be stored on a different medium, such as a different (e.g., less expensive) flash memory chip or hard drive. For non-memory mapped firmware 220, the hardware mechanism enabling direct processor access may be omitted. An important benefit of this approach is that the non-memory mapped firmware source may be cheaper, since the additional complex hardware associated with memory mapping may be omitted. Non-memory mapped firmware 220 is executed from a logical address space (e.g., pages in region 236 of CAR 230).

[0015] System 200 includes a paging handler 215, which may reside on memory mapped firmware, or may be retrieved from non-memory mapped firmware 220 and put into a reserved portion of CAR 230 for use during the execution phase (see FIG. 3). Paging handler 215 utilizes and controls CAR region 230 to implement a paging mechanism. For example, CAR region 230 may be partitioned to include a data region 232, a page table region 234, and a page region 236. Page region 236 may be configured to store an appropriate number

of pages. The minimum number of pages may be dependent on the nature of the instructions used by the processor, while the maximum number of pages may be dependent on the size of CAR region 230. For example, page region 236 may be configured to store at least three pages so that instructions on one page may be used to copy data between two other pages.

[0016] System 200 may be implemented in a number of ways. For example, system 200 may be implemented in an architecture in which a chipset is provided to support the microprocessor. Portions of system 200, such as memory mapped firmware 210 and/or non-memory mapped firmware 220 may be incorporated in a chipset separate from a microprocessor. System 200 may be included in a data processing system such as a computer or other data processing system. Of course, many configurations are possible.

[0017] FIG. 3 shows an implementation of a method 300 for executing system firmware. The microprocessor may first access the memory-mapped firmware (310). Although the memory-mapped firmware includes some of the instructions and data for initializing the system, other instructions and data are included in the non-memory mapped firmware.

[0018] For instructions and data included in the non-memory mapped firmware, the memory mapped firmware is directed to one or more pages of a CAR region or other memory region (320). If the desired data is available, the instruction is executed from the CAR region (330, 340). If the appropriate page (or pages) are not in the CAR region pages, a page fault processor exception is generated (330, 350). In response, a paging handler accesses the non-memory mapped firmware and loads the desired data into the CAR region (360). The paging handler then returns a fault serviced signal (370). The memory mapped firmware may then access the desired page(s) in the CAR region and execute the instruction (380). Portions of the process may repeat; for example, by checking to determine whether data is available (330) after accessing data. Further, after executing an instruction from the CAR (340), the page region of CAR may be accessed one or more additional times (320), and additional instructions may be executed.

[0019] The systems and techniques described above allow at least some instructions to be stored on non-memory mapped firmware. Since integrated circuits implementing non-memory mapped firmware need not include hardware for memory mapping, the firmware cost may be reduced. Additionally, enabling paging using a CAR region allows the firmware as a

whole to execute as though it is running from a contiguous memory region, when instead it is running from a limited page set.

[0020] The systems and techniques may additionally provide for faster system startup. In existing systems, firmware is typically executed uncached rather than from fast cache memory. Using system cache as RAM may also ease the difficulty of programming system startup by providing a region of memory that can be used early in the start up process. In existing systems, a substantial useable memory region is not available until the main memory is initialized. Since memory initialization generally involves executing a complex set of instructions, usable memory may not be available for a significant period. With CAR, the main memory initialization may also make use of the readable and writable memory region.

[0021] Having useable memory available early in the start-up process may also allow more efficient firmware storage by allowing data compression to be used. For example, the non-memory mapped firmware described above may include instructions or data which is compressed. The paging handler may include a decompression algorithm. The instructions may be efficiently stored using a compression algorithm. In response to a page fault exception for the

firmware instructions, the paging handler may implement a decompression algorithm to decompress the instructions into the CAR page. When the page fault handler completes, the decompressed instructions may subsequently be executed.

[0022] A number of implementations have been described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. For example, firmware may be stored on other types of memory than those discussed here. For example, at least some non-memory mapped firmware may be stored on a hard drive and brought to the CAR region using the paging mechanism. The paging mechanism may be implemented using a memory other than cache. Accordingly, other implementations are within the scope of the following claims.